

最も多くの石を裏返らせる手

Phase1

# 実装のポイント

ThinkingEngine.javaのthink(Board currentBoard)メソッドを修正します。

```
J ThinkingEngine.java ×
J ThinkingEngine.java > ThinkingEngine > chooseRandomMove(List<String>)
8 public class ThinkingEngine {
11
12     /**
13      * 現在の盤面情報に基づいて最適な着手を決定する。
14      * @param currentBoard 現在の盤面オブジェクト
15      * @return 着手文字列 (例: "c5" または "pass")
16      */
17     public String think(Board currentBoard) {
18
19         int myColor = currentBoard.getMyColor();
20
21         // 1. 合法的な着手リストを取得
22         List<String> legalMoves = currentBoard.getLegalMoves(myColor);
23
24         // 2. 合法手が一つもない場合
25         if (legalMoves.isEmpty()) {
26             return "pass";
27         }
28
29         // -----
30         // 【★ 書き換えるべき思考ロジックの領域 ★】
31         // -----
32
33         // 現状：ランダムに合法手を選ぶ (最弱AI)
34         String bestMove = chooseRandomMove(legalMoves);
35
36         // -----
37         // TODO: TODO
38         // 1. 評価関数 (Evaluation function) を実装する。
39         // 2. ミニマックス法 (Minimax) やα-β法 (Alpha-Beta Pruning) を実装する。
40         // 3. 探索深さを決定し、評価関数に基づき最適な手を返すように書き換える。
41         // -----
42
43         return bestMove;
44     }
```

# Step1. 合法手のリストアップ

Board.javaのgetLegalMoves(myColor)を使って、着手可能な座標のリストを取得します。(これは既にサンプルコードに実装されています。)



## Step2. 評価ループの実装

合法手リストの各手について、以下の処理を行うループを作成します。

- 着手のシミュレーション: その手を打った場合に、**何個の石が裏返るか** を正確に計算する必要があります。

ヒント: Board.javaには、着手後の盤面を返すようなメソッドがないため、このシミュレーションロジックをThinkingEngine内に追加する必要があります。

- 最大値の追跡: 現時点での裏返し最大数と、その最大数を達成した着手を記録します。

## Step3.着手の選択

ループが完了した後、記録した「裏返し最大数」を達成した着手を返します。

```
// ThinkingEngine.java (thinkメソッド内)
```

```
public String think(Board currentBoard) {
    List<String> legalMoves = currentBoard.getLegalMoves(myColor);
    if (legalMoves.isEmpty()) return "pass";

    String bestMove = legalMoves.get(0); // 仮の最善手
    int maxFlips = -1;

    for (String move : legalMoves) {
        // 1. 着手を座標(r, c)に変換
        // 2. moveを打った場合の裏返し数 (flips) を計算するロジックを実装
        //    * 注: このロジックは複雑なため、Boardクラスに「裏返し数を計算するメソッド」を
        //        追加することを検討してください。

        int currentFlips = calculateFlips(currentBoard, move, myColor); // ★実装が必要な関数

        if (currentFlips > maxFlips) {
            maxFlips = currentFlips;
            bestMove = move;
        }
    }

    return bestMove; // 最も裏返し数が多い手
}
```

## ヒント1: currentBoard

縦x横の二次元配列になっています。

黒石は1、白石は2となっています。

右のcurrentBoardはゲーム開始時点のボードを表しています。

4行目の4列目が白石、

4行目の5列目が黒石です。

```
▼ 変数
  ▼ Local
    ▼ currentBoard = Board@10
      ▼ cells = int[8][]@43
        > 0 = int[8]@45
        > 1 = int[8]@46
        > 2 = int[8]@47
        ▼ 3 = int[8]@48
          0 = 0
          1 = 0
          2 = 0
          3 = 2
          4 = 1
          5 = 0
          6 = 0
          7 = 0
        > 4 = int[8]@49
        > 5 = int[8]@50
        > 6 = int[8]@51
        > 7 = int[8]@52
```

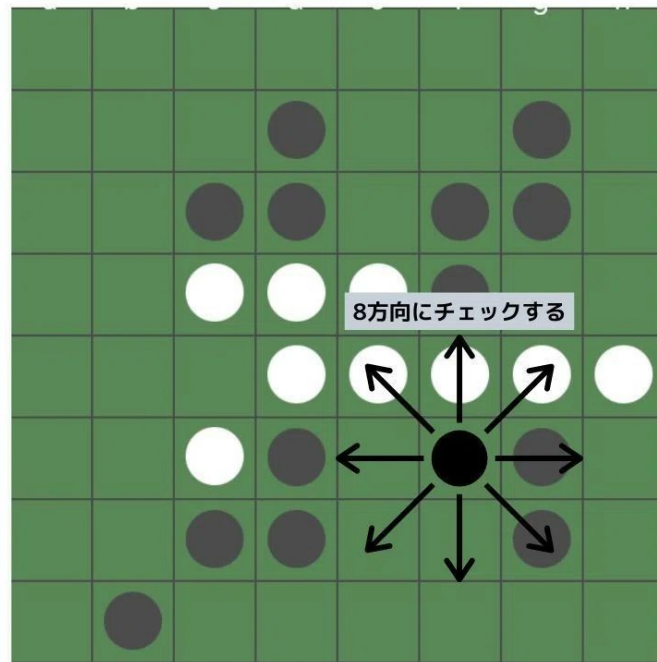
## ヒント2:8方向にチェックする

石を置いた時に何個裏返るかを計算する際には8方向すべてをチェックする必要があります。

このロジックは既にサンプルプログラムの中に実装されています。

指定座標が合法手であるかどうかを判定する

Board.javaのisLeagalMove()関数です。



# Board.java isLegalMove()

row=行、col=列です。  
(colはcolumnの略)

drとdcは行と列の方向です。

左上 : dr[0]=-1, dc[0]=-1  
上 : dr[1]=-1, dc[1]= 0  
右上 : dr[2]=-1, dc[2]= 1  
左 : dr[3]= 0, dc[3]=-1  
右 : dr[4]= 0, dc[4]= 1  
左下 : dr[5]= 1, dc[5]=-1  
下 : dr[6]= 1, dc[6]= 0  
右下 : dr[7]= 1, dc[7]= 1

```
/** 座標が合法手であるかを判定 (着手色を指定) */
public boolean isLegalMove(int row, int col, int color) {
    if (row < 0 || row >= SIZE || col < 0 || col >= SIZE || cells[row][col] != EMPTY) {
        return false;
    }

    int opponent = (color == BLACK) ? WHITE : BLACK;
    int[] dr = {-1, -1, -1, 0, 0, 1, 1, 1};
    int[] dc = {-1, 0, 1, -1, 1, -1, 0, 1};

    for (int i = 0; i < 8; i++) {
        int r = row + dr[i];
        int c = col + dc[i];
        boolean foundOpponent = false;

        while (r >= 0 && r < SIZE && c >= 0 && c < SIZE && cells[r][c] == opponent) {
            foundOpponent = true;
            r += dr[i];
            c += dc[i];
        }

        if (foundOpponent && r >= 0 && r < SIZE && c >= 0 && c < SIZE && cells[r][c] == color) {
            return true;
        }
    }
    return false;
}
```